AI ENGINEERING & PRODUCT

Базовые термины LLM и AI-систем

Мини-лекция 10–15 минут

Для инженеров и продакт-менеджеров

Основы | Архитектура | Производительность | Безопасность

Александр Штаченко // ProGameDev.net

2026-03-18 • v1.0

Как работает LLM: ключевые понятия

Токен (Token)

Минимальная единица текста для модели (слово, часть слова или символ). Лимиты контекста и стоимость API рассчитываются именно в токенах.

Контекст (Context)

Всё, что модель «видит» в момент генерации: системные инструкции, история диалога, загруженные файлы и результаты вызова инструментов.

Промт (Prompt)

Текущий запрос пользователя к модели. В продакшене это лишь верхушка айсберга общего контекста, передаваемого в LLM.

Инференс vs Тренировка

Инференс: получение ответа от готовой модели (быстро, онлайн).
Тренировка: процесс обучения с обновлением весов (дорого, долго, офлайн).

Transformer: архитектура по умолчанию для LLM

Источник: "Attention Is All You Need" (Vaswani et al., 2017) / Wikimedia Commons

Золотой стандарт AI

Архитектура, вытеснившая RNN/LSTM. Ключевая инновация — механизм **Self-Attention**, позволяющий параллельно обрабатывать весь контекст.

Параллельная обработка

В отличие от последовательных моделей, Трансформер может «смотреть» на все слова в предложении одновременно, что критично для масштабирования.

Дальние зависимости

Эффективно связывает слова, находящиеся далеко друг от друга в тексте (контекст), не теряя смысл при генерации длинных ответов.

Базовые термины LLM и AI-систем

Основные блоки

Attention (внимание), Feed-Forward Networks (полносвязные слои), Residual Connections (остаточные связи) и Layer Normalization.

💡 GPT (Generative Pre-trained Transformer) использует только **Decoder**-часть оригинальной архитектуры для авто-регрессионной генерации (токен за токеном).

Attention и Multi-head: суть и стоимость

Источник: Wikimedia Commons (Self-Attention Diagram)

Механизм Attention

Позволяет модели фокусироваться на релевантных частях контекста при генерации каждого токена. Это «память» модели в моменте.

Multi-head Attention

Несколько параллельных механизмов внимания. Одна «голова» следит за грамматикой, другая — за смыслом, третья — за фактами.

Квадратичная сложность $O(n^2)$

Вычислительная стоимость растёт как квадрат длины контекста. Удвоение контекста (4k → 8k) увеличивает нагрузку в ~4 раза.

⚠ Именно из-за $O(n^2)$ длинный контекст (128k+) резко увеличивает **стоимость** (API pricing) и **задержку** (latency) обработки.

Инференс: Prefill vs Decode

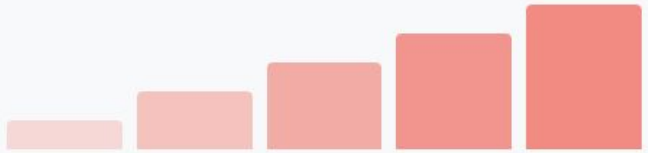
Prefill

- **Обработка контекста:** Модель «проглатывает» весь входной промт и историю целиком за один раз.
- **Параллельность:** Отлично использует мощность GPU (матричные операции), поэтому выполняется *быстро*.
- **Зависимость:** Время растёт с увеличением длины *входного* (input) текста.

VS

Decode

- **Генерация ответа:** Последовательное создание по *одному токену* за шаг (авто-регрессия).
- **Последовательность:** Нельзя распараллелить (следующий токен зависит от предыдущего) → *медленно*.
- **Зависимость:** Время линейно растёт с длиной *выходного* (output) ответа.



AI-систем

KV-Cache и Latency: ускорение генерации

Что такое KV-Cache?

Кэш промежуточных вычислений (Keys/Values) слоев attention для уже обработанных токенов. Позволяет не пересчитывать весь контекст заново на каждом шаге генерации.

Снижение задержки (Latency)

Критически ускоряет фазу **Decode** (генерацию токенов). Без кэша сложность генерации росла бы квадратично от длины ответа, с кэшем — линейно.

Важность для диалогов и агентов

В многошаговых сценариях (chat, agent loops) история растет. KV-кэш позволяет сохранять "состояние ума" модели между репликами без повторного тяжелого префилла.

Цена вопроса: Память GPU

KV-кэш занимает значительный объем видеопамяти (VRAM). При длинном контексте размер кэша может превышать размер весов самой модели.

Экономика токенов и кэширование

Input vs Output tokens

Input: всё, что отправлено в модель (промт + контекст).
Output: всё, что сгенерировано. Тарификация обычно раздельная, output часто дороже.

Кэширование (Caching)

Повторное использование ранее обработанных данных (KV-cache) для снижения стоимости и задержки (latency). Платите только за новый контекст.

Оптимизация контекста

Сокращайте объем передаваемых данных: используйте тримминг истории, сжатие контекста и саммаризацию для уменьшения input tokens.

Latency vs Cost Tradeoff

Баланс между скоростью и стоимостью: кэширование снижает и то, и другое, но требует управления состоянием (stateful).

Управление генерацией: параметры выборки

Temperature (Температура)

Параметр случайности: чем выше (например, 0.8–1.0), тем более вариативный и креативный ответ. Низкая температура (0.0–0.2) делает результат предсказуемым и детерминированным.

Top-P (Nucleus Sampling)

Ограничение выборки токенов по суммарной вероятности (например, 0.9). Модель рассматривает только самые вероятные варианты, составляющие "ядро" вероятности.

Top-K

Жесткое ограничение на фиксированное количество возможных токенов (например, K=50). Отсекает "хвост" маловероятных слов, предотвращая бессмыслицу.

Практические рекомендации

Для фактов, кода и извлечения данных используйте **низкую temperature** (~0). Для креатива и идей — **высокую temperature** + Top-P для баланса разнообразия и связности.

Архитектура: Stateless vs Stateful

Stateless

- **Без памяти:** Каждый запрос «как новый», система ничего не помнит о предыдущих шагах.
- **Простота:** Легко масштабировать, нет риска рассинхронизации данных.
- **Применение:** Одношаговые задачи (перевод, суммаризация, классификация).

СХЕМА РАБОТЫ

VS

Stateful

- **С состоянием:** Хранит контекст, историю шагов, план действий и результаты.
- **Агенты:** Критично для многошаговых сценариев и сложных диалогов.
- **Сложность:** Требует управления памятью (CRUD), прав доступа и очистки state.

СХЕМА РАБОТЫ

Агент и цикл Agent Loop

Цикл работы агента: Восприятие → Рассуждение → Действие

Что такое Агент?

LLM + память + инструменты + способность принимать решения. Это система, которая не просто отвечает, но и выполняет задачи.

Agent Loop (Цикл)

Бесконечный или итеративный процесс: **Observe** (получить данные) → **Reason** (понять, что делать) → **Act** (вызвать инструмент).

Отличие от Чат-бота

Бот пассивно отвечает на текст. Агент активно меняет среду: пишет в БД, отправляет API-запросы, запускает код.

Безопасность

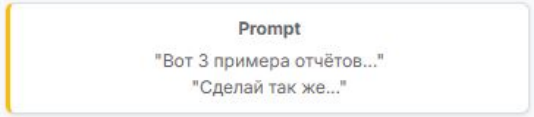
Требуются строгие права доступа (sandbox), аудит действий и "стоп-контуры" (human-in-the-loop), чтобы агент не натворил бед.

⚠ Автономность агента — главный риск. Всегда ограничивайте количество шагов и доступные инструменты.

Данные: In-context vs RAG

In-context Learning

- **Принцип:** Передача знаний прямо в промте (инструкции, примеры few-shot).
- **Плюсы:** Мгновенно, не требует сложной инфраструктуры, легко тестировать.
- **Минусы:** Ограничено длиной контекста (token limit), дорого на каждый запрос.



Вся информация внутри запроса

VS

RAG (Retrieval-Augmented)

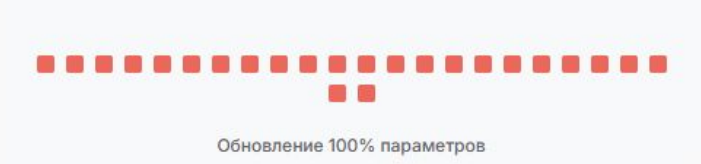
- **Принцип:** Поиск релевантных фактов во внешней базе *перед* генерацией ответа.
- **Компоненты:** Embeddings (векторы смысла) + Vector Database (хранилище).
- **Плюсы:** Обновляемые знания без переобучения модели, работа с гигабайтами данных.

Retrieval → Augment Context → Generation

Настройка модели: Fine-tuning vs LoRA

Full Fine-tuning

- **Полное обновление:** Изменяются веса всей модели (миллиарды параметров).
- **Ресурсоемко:** Требует мощных GPU кластеров и сложной MLOps инфраструктуры.
- **Стабильность:** Фундаментальное изменение поведения, но риск «забывания».

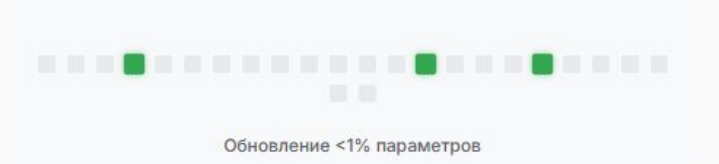


Обновление 100% параметров

VS

LoRA (PEFT)

- **Адаптеры:** База заморожена, обучаются лишь малые слои (<1% параметров).
- **Быстро и дешево:** Возможно обучение на одной карте (даже потребительской).
- **Модульность:** Легко переключать адаптеры (стили, форматы) на лету.



Обновление <1% параметров

Context Engineering: что важнее промта

Управление всем контекстом

Проектирование и оркестрация всех компонентов: памяти, retrieval (RAG), текущего состояния (state), доступных инструментов и политик безопасности.

Dynamic Prompt Assembly

Динамическая сборка финального запроса к модели «на лету» из разных источников данных (профиль пользователя, найденные документы, результаты предыдущих шагов).

Prompt Engineering — лишь часть

Оптимизация формулировок (промт-инжиниринг) — это подмножество context engineering. Качество системы в продакшене чаще зависит от правильной сборки контекста, чем от «магических слов».

Память (Memory)

Хранение и извлечение фактов о пользователе или проекте. Важные аспекты: срок жизни памяти, релевантность выборки и приватность данных.

Безопасность и метрики

Безопасность (Security)

Prompt Injection

Атака, при которой пользовательский ввод заставляет модель игнорировать системные инструкции. Защита: валидация ввода/вывода, разделение данных и инструкций.

Sandbox (Песочница)

Изолированная среда для безопасного выполнения кода и вызова инструментов. Принцип минимальных привилегий для агентов.

Shadow AI & Backdoor

Риски использования неконтролируемых AI-инструментов сотрудниками и скрытое вредоносное поведение в моделях.

Метрики (Metrics)

Throughput (Пропускная способность)

Количество запросов или токенов, обрабатываемых системой за единицу времени (RPS / tokens per second).

Scalability (Масштабируемость)

Способность системы эффективно расти под нагрузкой без деградации качества и задержек.

Latency vs Cost Tradeoff

Компромисс: ускорение ответа часто стоит дороже (мощнее GPU, меньше сжатие), а снижение цены увеличивает задержку.